WEST

Freeform Search

Database:
US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: L1 and (@ad<20000918 or @rlad<20000918 or @prad<20000918)

Display: 40 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ Hit List ◉ Hit Count ○ Side by Side ○ Image

Search | Clear | Help | Logout | Interrupt

Main Menu | Show S Numbers | Edit S Numbers | Preferences | Cases

Search History

DATE: Thursday, September 25, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
L2	L1 and ((@ad<20000918 or @rlad<20000918 or @prad<20000918)	134	L2 Searched all
L1	((common configuration?) or registry) same (file? or database?) and redirect$3	204	L1

END OF SEARCH HISTORY

WEST

Help Logout Interrupt

Main Menu | Search Form | Posting Counts | Show S Numbers | Edit S Numbers | Preferences | Cases

Search Results -

Terms	Documents
(operating system?) and (redirect$ or (re direct$)) and bit and (@ad<20000918 or @rlad<20000918 or @prad<20000918)	955

Database:
US Patents Full-Text Database
US Pre-Grant Publication Full-Text Database
JPO Abstracts Database
EPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L7 not (16 or 12 or 13)

Refine Search

Recall Text Clear

Search History

DATE: Thursday, May 15, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=USPT,PGPB,JPAB,EPAB,DWPI,TDBD; PLUR=YES; OP=ADJ		
L7	(operating system?) and (redirect$ or (re direct$)) and bit and (@ad<20000918 or @rlad<20000918 or @prad<20000918)	955	L7
L6	L5 and (@ad<20000918 or @rlad<20000918 or @prad<20000918)	198	L6 Scanned all
L5	(redirect$ or (re direct$)) and registry and (node? or loci or locus)	296	L5
L4	L1 and (locus.ti. or loci.ti. or locus.ti. or locus.ab)	0	L4
L3	L1 and (registry.ti. or registry.ab)	1	L3 Scanned all
L2	L1 and (redirect$3.ti. or redirect$3.ab)	12	L2 Scanned all
L1	redirect$3 and regist$5 and (node? or loci or locus)	1621	L1

END OF SEARCH HISTORY

WEST

Generate Collection | Print

L7: Entry 515 of 955 File: USPT Aug 15, 2000

DOCUMENT-IDENTIFIER: US 6105101 A
TITLE: 16 bit bios interrupt calls under 32 bit protected mode application

Abstract Text (1):
A method for performing 16 Bit BIOS interrupt calls under a 32 Bit protected mode application. This has been impossible to-date and has forced BIOS development teams to add support into the BIOS for 32 bit function calls from 32 bit applications.

Application Filing Date (1):
19980506

Brief Summary Text (2):
This application relates to computer operating systems, and more particularly to 16-bit interrupt calls within a 32-bit operating system.

Brief Summary Text (3):
BACKGROUND: COMPUTER OPERATING SYSTEMS

Brief Summary Text (5):
Operating systems have grown increasingly complex over time. One change has been a gradual move from 8-bit, to 16-bit, to 32-bit operating systems. In order to facilitate backwards compatibility, it is preferable to provide means for applications written for, e.g., a 16-bit operating system to function in other operating systems, such as a 32-bit system.

Brief Summary Text (7):
Modern operating systems are typically built in layers, with each layer adding new capabilities, such as disk access techniques or a graphical user interface. But the essential layer, the foundation on which the rest of the operating system rests, is typically called a kernel. In general, the kernel provides low-level services, such as memory management, basic hardware interaction, and security. Without the kernel, the system would stop.

Brief Summary Text (11):
The current method used to call 16-bit BIOS from an application running in 32-bit operating system is to pass the data to a 16-bit real mode data server via OLE automation, and let the 16-bit real mode data server perform the interrupt. The process of converting from a 32-bit mode to a 16-bit mode is called "thunking," and results in significant wasted processor time. If an error occurs the results are unpredictable. The increased overhead decreases performance of the system.

Brief Summary Text (12):
The term "thunk" comes from the hypothetical sound a system process such as a Dynamic Linking Library (DLL) might make when switching between 16-bit and 32-bit operation. To call a 16-bit function with a 32-bit application, the application must first build a 16-bit stack and perform numerous other setup tasks, then call a thunk function.

Brief Summary Text (13):
A limitation of current methods is that while it is possible to call a 16-bit function with a 32-bit application by thunking, it is not possible to fully perform a 16-bit interrupt call by a 32-bit application.

Brief Summary Text (15):
The general purpose of the invention is to provide a mechanism of performing 16 Bit BIOS interrupt calls under a 32 Bit protected mode application. This has been impossible to-date and has forced the BIOS development team to add support into the BIOS for 32 bit function calls from 32 bit applications.

Brief Summary Text (16):
The preferred method passes the interrupt request to a 32 Bit protected mode driver. The driver will construct the 16 Bit call, change the stack to a 16 Bit stack, and make the 16 Bit protected mode BIOS call. This process is much faster than the previous method. Because there is no mode switch (Thunk) from protected to real and from real back to protected mode, the process of making the ROM call is not noticeable. Also, this process only requires that the BIOS write 16-bit mode code that is capable of running in real or protected mode. This is accomplished by not hard-coding F000h as a data segment/selector. There is no need to re-code BIOS to support 32 Bit calls, which saves space and time.

Detailed Description Text (3):
The preferred embodiment provides a method for passing a 16-bit interrupt call in a 32-bit operating system. This method avoids the cost of thunking, and allows the calls to be made virtually invisibly to the application programmer.

Detailed Description Text (4):
The method is preferably embodied in a Windows NT 3.51/4.0 device driver. This driver is loaded when the NT system is started, and will automatically perform the preferred method when a 16-bit interrupt call is made.

Detailed Description Text (6):
(a.) When an application requires a 16-bit interrupt, it makes a call to a "Do16BitInterrupt" function (step 210), and passes the necessary arguments as described in more detail below.

Detailed Description Text (7):
(b.) Upon detection of a request to make a 16-bit interrupt call, the "clntmgmt.sys" device driver creates a buffer in memory to hold the remapping code (step 220).

Detailed Description Text (11):
(f.) Using the Ke386Do16BitFunction call (step 260), the device driver makes the call to the remapper, which in turn redirects the call to BIOS.

Detailed Description Text (14):
Note that 16-bit functions and applications use a "segment:offset" addressing scheme. "40:0" therefore indicates an address at segment 40, offset 0. In typical x86 computer systems, the 40:0 area is the BIOS data area. A selector is a pointer to data structures that define the characteristics of a memory segment.

Detailed Description Text (15):
Using the process described above, the application is able to simply call the Do16BitInterrupt function, and the call is made without any requirement of manually converting to the 16-stack to make the call. Windows NT 4.0 provides one undocumented 16-bit function, called KeI386Do16BitFunction, which is used for Advanced Power Management (APM) functions. However, Kei386Do16BitFunction does not provide for IRET returns used by interrupts, and is therefore unsuitable for 16-bit interrupt applications.

Detailed Description Text (17):
Therefore, when a 16-bit interrupt is needed, the NT Kernel sends the 16-bit function to the remapper, instead of directly to ROM (for which the user would manually have to switch (thunk) to 16 bits). The remapper changes the stack, and pushes our function on the stack. The call is passed to ROM with the values on the Stack. ROM executes the 16-bit call, adds the result to the stack and passes it back to the remapper. The remapper restores the stack, and passes the result back to the kernel.

CLAIMS:

4. The method of claim 1, wherein said computer system is operating in a 32-bit mode.

5. The method of claim 1, wherein said remapped interrupt request is a 16-bit interrupt.

9. The method of claim 6, wherein said first mode is a 32-bit operating mode.

10. The method of claim 6, wherein said second mode is a 16-bit operating mode.

14. The system of claim 11, wherein said first mode is a 32-bit operating mode.

15. The system of claim 11, wherein said second mode is a 16-bit operating mode.

WEST

Generate Collection Print

L7: Entry 863 of 955 File: USPT Jun 27, 1995

DOCUMENT-IDENTIFIER: US 5428799 A
TITLE: Redirection of interrupts to microprocessors

Abstract Text (1):
A redirector in either software or hardware can is used to distribute global interrupts among all processors of a multiprocessor (MP) system, which may have multiple architecturally-isolated buses. When redirecting interrupts using software, a default processor receives all interrupts and redirects them to destination processors. When redirecting interrupts using hardware, a discrete redirector in hardware forwards all interrupts to particular processors based on a routing, or look-up, table. Another implementation incorporates interrupt snooping with either software or hardware interrupt redirection where interrupt response is critical for I/O cards.

Application Filing Date (1):
19940527

Brief Summary Text (23):
The present invention envisions implementing a redirector which receives global interrupts from a system interrupt controller. The redirector may be either in the form of software in a default processor or a discrete component of hardware, such as a programmable logic array or arrays.

Brief Summary Text (24):
After receiving global interrupts, the redirector directly notifies individual processors in the MP system of the global interrupts. Some of the individual processors could exist on remote, architecturally-isolated buses, perhaps on input/output (I/O) cards.

Drawing Description Text (6):
FIG. 4 shows a preferred embodiment of the present invention which implements a redirector, in the form of hardware, software, or a combination thereof, for channelling global system interrupts from the system interrupt controller directly to individual CPUs of the MP system;

Drawing Description Text (8):
FIG. 5(a) shows the methodology for implementing software (S/W) interrupt redirection;

Drawing Description Text (9):
FIG. 5(b) shows the methodology for implementing hardware (H/W) interrupt redirection; and

Drawing Description Text (10):
FIGS. 6 in combination with FIG. 7 shows an example of a specific hardware implementation of the redirector of the present invention, wherein FIG. 6 illustrates a vector translation sender situated locally to the system interrupt controller and FIG. 7 illustrates a vector translation receiver situated locally to each of the n MPUs of the MP system.

Detailed Description Text (21):
FIG. 4 shows the preferred embodiment of the present invention. A redirector 402 is

implemented to direct interrupts from the system interrupt controller 130 to individual CPUs on any bus, including individual CPUs on an I/O card, for instance, I/O card 126 or 128.

Detailed Description Text (22):
As shown, the redirector 402 is communicates with the system interrupt controller 130 via a bus 404. Moreover, the redirector 402 communicates with the individual n MPUs 102-108.

Detailed Description Text (23):
The redirector 402 may be optimally implemented in predominantly software, predominantly hardware, or a combination of software and hardware. In the predominantly software implementation, the functionality of the redirector 402 is implemented by program code executed by an arbitrary CPU, sort of a "default CPU", but with a different function, as described below in FIG. 5(a).

Detailed Description Text (24):
In the predominantly hardware implementation, the functionality of the redirector 402 is effectuated by physical logic circuitry. The logic circuitry can be implanted on an external microchip, such as an application specific integrated circuit (ASIC), or any other discrete circuit apparatus(es). Further, the system interrupt controller 130 and the redirector 402 may be incorporated on the same circuit apparatus, such as an ASIC. In the preferred embodiment, programmable logic arrays (PAL) are utilized to implement the hardware logic.

Detailed Description Text (26):
With reference to FIGS. 5a and 5b, the methodology of system interrupt redirection in either software or hardware is described below. FIG. 5(a) pertains to software (S/W) interrupt redirection, whereas FIG. 5(b) pertains to hardware (H/W) interrupt redirection.

Detailed Description Text (27):
(1) At flowchart blocks 514 and 524, either (a) the redirector 402 or (b) the default CPU having the code corresponding to the function of redirector 402 receives the global system interrupt from the system interrupt controller 130.

Detailed Description Text (28):
(2) At flowchart blocks 516 and 526, either (a) the redirector 402 or (b) default CPU, using a look-up table, finds the destination CPU and its associated I/O address, the corresponding interrupt vector, and the interrupt's priority.

Detailed Description Text (29):
(3) At flowchart block 518 and 528, either (a) the redirector 402 or (b) the default CPU then writes the contents of the look-up table to the destination CPU.

Detailed Description Text (30):
Worth noting is that the present invention does not require that the redirector 402 or the default CPU get control of, or "own," the destination CPU, such as in semaphore type systems, before sending an interrupt. Semaphores indicate whether or not the CPU is ready or capable of receiving an interrupt. They insure that a CPU is not overwhelmed at one time with multiple interrupt commands.

Detailed Description Text (31):
The reason that neither the default CPU in the software implementation nor the redirector 402 in the hardware implementation need not own the destination CPU is that only one potential source of global interrupts exists in the MP system. Moreover, the destination CPU handshakes or acknowledges the interrupt from the default CPU or redirector 402.

Detailed Description Text (32):
A further feature of the present invention is that the redirector 402 serves as sort of a buffer/interface between the MPUs 110-116 of the MP system. In other words, the present invention permits homogeneous and nonhomogeneous processors to operate from the same system interrupt controller 130.

Detailed Description Text (34):
The functionality of the redirector 402 is implemented partly at a distinct central location, possibly at or near the system interrupt controller 130, and partly at a location which is local to each of the n MPUs 102-108. FIG. 6 illustrates that part which is associated with the system interrupt controller and which is called a vector translation sender 602, whereas FIG. 7 shows that part which is associated with each individual MPU and which is called a vector translation receiver 702. Essentially, the vector translation sender channels system interrupts to specific CPUs of the n MPUs 102-108, while the vector translation receiver comprises handshaking logic and logic for servicing local interrupts.

Detailed Description Text (37):
The vector translation sender operates as follows. A 16-bit system interrupt (IRQ 15:0) from somewhere in the MP system enters a latch 608, which is under the control of a PAL1 610. The PAL1 610 controls the latch 608 via a chip enable (CE) control 612 and a direction (DIR) control 614. Moreover, the latch 608 is comprised of a set of parallel PALs 616-622.

Detailed Description Text (38):
The cascaded "INTEL" 8259 interrupt controllers 604 and 606 (system interrupt controller 130) receive the 16-bit system interrupt via a 4-bit bus 624 (I15:I12), a 4-bit bus 626 (I11:I8), a 4-bit bus 628 (I7:I4), and a 4-bit bus 630 (I3:I0). Subsequently, the "INTEL" 8259 interrrupt controllers 604 and 606 notify the a PAL1 610 of the system interrupt via an interrupt line 632. The PAL1 610 then forwards an interrupt acknowledge to the cascaded "INTEL" 8259 interrupt controllers 604 and 606 via an interrupt acknowledge (INT ACK) line 634.

Detailed Description Text (39):
Next, the set of cascaded "INTEL" 8259 interrupt controllers 604 and 606 (system interrupt controller 130) forward a 4-bit interrupt vector (vd3:0), which could designate any of 16 different kinds of system interrupts, to the PAL1 610 and a PAL3 636. Accordingly, the PAL1 610 freezes the latch 608 via the chip enable control 612 and further enables a random access memory (RAM) 638 via a chip enable control 640 and a direction control 642.

Detailed Description Text (40):
The PAL3 636 translates the 4-bit interrupt vector into an address (va3:va0) which is sent to a look-up table in the RAM 638. Consequently, a 16-bit data word is transmitted onto the system data bus 644 from the RAM 638, as shown by a bus 646. The system data bus 644 could be a combination of several buses. Further, in the preferred embodiment, the 16-bit data word comprises an interrupt vector (8 bits), a vector priority, and the address (slot number) of the CPU to receive the interrupt vector.

Detailed Description Text (41):
At the same time that the foregoing 16-bit data word is outputted by the RAM 638, PAL3 636 generates an I/O cycle on the system control bus 648 via controls 650 and 652 in order to write the 16-bit data word to the destination CPU. The system control bus 648 comprises both addressing and control lines of the MP system and also may be a combination of buses. As shown, the PAL3 636 has the ability to implement the following commands on the system control bus 648: (1) a control signal indicating either command or data (CMD/D), (2) a control signal indicating either memory or I/O (M/IO), (3) a control signal indicating either read or write (RD/WR), (4) a bus request (BUSRQ) signal, and (5) a bus acknowledge (BUSACK) signal.

Detailed Description Text (44):
When an external I/O device attempts to access the cascaded "INTEL" 8259 interrupt controllers 604 and 608, a PAL4 658 and a PAL5 660 are enabled to permit access to the "INTEL" 8259 interrupt controllers 604 and 606 from the system data bus 644. In contrast, when a system interrupt is being sent to one of the n MPUs 102-108, the PAL4 658 and PAL5 660 are disabled to isolate the data lines (vd7:0) 654 and 656 from the system data bus 644 and, thus, from the 16-bit data word ultimately outputted by the RAM 638.

Detailed Description Text (47):

An single "INTEL" 8259 interrupt controller 716 receives the 7-bit interrupt vector (I7:I0) through parallel buses 718 (I7:I4) and 720 (I3:0) as shown. The "INTEL" 8259 interrupt controller 716 prioritizes the 7-bit interrupt vector (along with any local interrupts) and translates the vector into a 3-bit interrupt vector (vd2:0).

Detailed Description Text (50):
The "INTEL" 8259 interrupt controller 716 forwards a 3-bit interrupt vector (vd2:0), which could designate any of 8 different kinds of interrupts, to the PAL1 710. Accordingly, the PAL1 710 freezes the latch 704 via the cache enable control 712 and enables a RAM 738 via a cache enable control 740. Moreover, the PAL1 710 translates the 3-bit interrupt vector into an address (va2:va0) which is sent to a look-up table in the RAM 738. Consequently, the original interrupt vector (I7:I0) retrieved from the system data bus 644, which could be any of 8 different possibilities, is used to identify another vector, which can be any of 256 different possibilities.

Detailed Description Text (54):
The system and method of the present invention may also be implemented with interrupt snooping, as discussed above. The implementation would be a hybrid between interrupt snooping and either software interrupt redirection or hardware interrupt redirection. Such an approach might be desirable in an MP system having I/O cards with critical response needs.

Detailed Description Text (55):
The software interrupt redirection is preferred in PCs having only 1, 2, or 3 processors and associated subsystems. One reason is that the present invention can be implemented at low cost. Another reason is that it does not burden the MP system with additional hardware. Still another reason is that the software overhead is not great. Finally, the system and method can more easily be designed to interface across multiple architecturally-isolated buses.

Detailed Description Text (56):
The hardware interrupt redirection is a more desirable form of the present invention in MP systems with a large number of global interrupts. In other words, the additional software overhead at the default CPU in such an MP system for servicing interrupts may grind the default CPU down to a halt.

Detailed Description Text (57):
Finally, software interrupt redirection, interrupt redirection, or interrupt snooping with either software or hardware interrupt redirection is compatible with the OS/2 or UNIX operating systems (drivers). The reason is that the global interrupts are transparent to the operating systems and application programs.

Related Application Filing Date (1):
19910213

CLAIMS:

1. A system for increasing the performance of a multiprocessor computer system, comprising:

a shared memory;

two or more processors coupled to said shared memory by a system bus, each of said two or more processors configured to transmit global interrupts to another two or more processors via said system bus;

a system interrupt controller, coupled to said system bus, configured to receive and retransmit all of said global interrupts transmitted by said two or more processors; and

a redirector, coupled to said system bus, configured to communicate with said system interrupt controller and said two or more processors, said redirector configured to receive all of said global interrupts retransmitted by said system interrupt controller, said redirector further configured to transmit interrupt data associated with said retransmitted global interrupts directly to a destination processor of

said two or more processors.

2. The system of claim 1, wherein said redirector and said system interrupt controller comprise:

one or more program interrupt controllers; and

software operated upon by a default processor of said two or more processors of the multiprocessor computer system.

5. The system of claim 1, wherein said system interrupt controller and said redirector are incorporated on an ASIC.

6. The system of claim 1, wherein said redirector comprises a vector translation sender corresponding to said system interrupt controller and a vector translation receiver corresponding to each of said two or more processors.

9. A method for increasing the performance of a multiprocessor computer system, the multiprocessor system having two or more processors, a system interrupt controller, and a redirector, the two or more processors, system interrupt controller, and redirector coupled to and communicating with each other via a system bus, the method comprising the steps of:

(1) receiving, in the system interrupt controller, a global interrupt from one of the two or more processors;

(2) determining, at the system interrupt controller, the priority of said global interrupt and an associated global interrupt vector;

(3) transmitting said global interrupt vector from the system interrupt controller to the redirector;

(4) determining, at the redirector, a system bus address of a destination processor of the two or more processors for receiving said global interrupt;

(5) translating said global interrupt into a destination processor dependent vector; and

(6) notifying said destination processor of said global interrupt by transmitting said destination processor dependent vector.

14. A system for increasing the performance of a multiprocessor computer system, comprising:

a shared memory;

two or more processors coupled to said shared memory by a system bus, each of said two or more processors configured to transmit global interrupts to another two or more processors via said system bus;

a system interrupt controller, coupled to said system bus, configured to receive and retransmit all of said global interrupts transmitted by said two or more processors, said system interrupt controller including,

one or more first programmable interrupt controllers coupled to said system bus,

a first discrete logic, coupled to said one or more first programmable interrupt controllers, implemented by a first plurality of programmable logic arrays, configured to control said system interrupt controller, and

a first memory unit, coupled to said discrete logic, configured to store said interrupt data; and

a redirector, coupled to said system bus, configured to communicate with said system interrupt controller and said two or more processors, said redirector configured to

receive all of said global interrupts retransmitted by said system interrupt controller, said redirector further configured to transmit interrupt data associated with said retransmitted global interrupts directly to a destination processor of said two or more processors, said redirector including,

one or more second programmable interrupt controllers coupled to said system bus,

a second discrete logic, coupled to said one or more second programmable interrupt controllers, implemented by a second plurality of programmable logic arrays, configured to control said redirector, and

a second memory unit, coupled to said second discrete logic, configured to store said interrupt data.